

# Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly  
language:

```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

Machine  
code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000011111101000011111
```

Computer  
system:



Memory & data  
Integers & floats  
**Machine code & C**  
x86 assembly  
Procedures & stacks  
Arrays & structs  
Memory & caches  
Processes  
Virtual memory  
Memory allocation  
Java vs. C

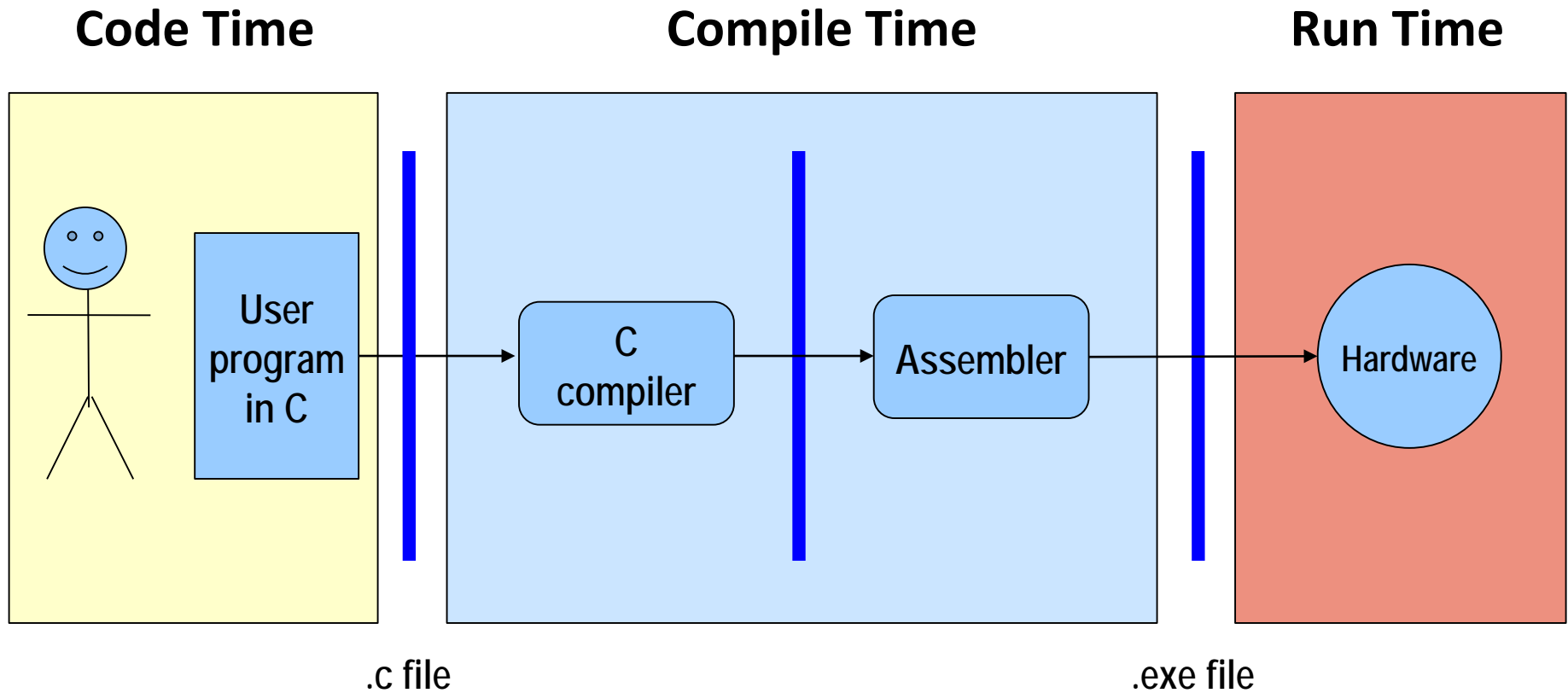
OS:



# Section 3: Basics of architecture, machine programming

- What is an ISA (Instruction Set Architecture)?
- A brief history of Intel processors and architectures
- C, assembly, machine code
- x86 basics: registers

# Translation



**What makes programs run fast?**

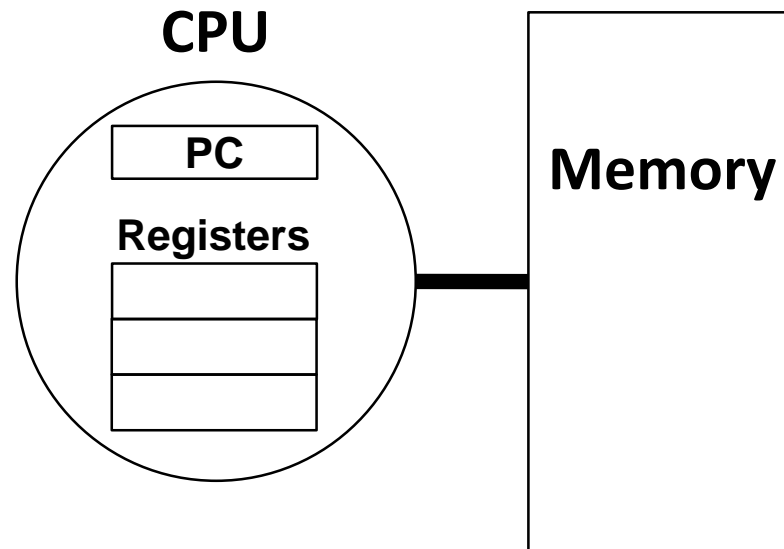
# Translation Impacts Performance

- **The time required to execute a program depends on:**
  - *The program* (as written in C, for instance)
  - *The compiler*: what set of assembler instructions it translates the C program into
  - *The instruction set architecture* (ISA): what set of instructions it makes available to the compiler
  - *The hardware implementation*: how much time it takes to execute an instruction

# Instruction Set Architectures

## ■ The ISA defines:

- The system's state (e.g. registers, memory, program counter)
- The instructions the CPU can execute
- The effect that each of these instructions will have on the system state



# General ISA Design Decisions

## ■ Instructions

- What instructions are available? What do they do?
- How are they encoded?

## ■ Registers

- How many registers are there?
- How wide are they?

## ■ Memory

- How do you specify a memory location?

# x86

- **Processors that implement the x86 ISA completely dominate the server, desktop and laptop markets**
- **Evolutionary design**
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on
- **Complex instruction set computer (CISC)**
  - Many different instructions with many different formats
    - But, only small subset encountered with Linux programs
  - (as opposed to Reduced Instruction Set Computers (RISC), which use simpler instructions)

# Intel x86 Evolution: Milestones

<i>Name</i>	<i>Date</i>	<i>Transistors</i>	<i>MHz</i>
■ <b>8086</b>	<b>1978</b>	<b>29K</b>	<b>5-10</b>
<ul style="list-style-type: none"> <li>▪ First 16-bit processor. Basis for IBM PC &amp; DOS</li> <li>▪ 1MB address space</li> </ul>			
■ <b>386</b>	<b>1985</b>	<b>275K</b>	<b>16-33</b>
<ul style="list-style-type: none"> <li>▪ First 32 bit processor, referred to as IA32</li> <li>▪ Added “flat addressing”</li> <li>▪ Capable of running Unix</li> <li>▪ 32-bit Linux/gcc targets i386 by default</li> </ul>			
■ <b>Pentium 4F</b>	<b>2005</b>	<b>230M</b>	<b>2800-3800</b>
<ul style="list-style-type: none"> <li>▪ First 64-bit Intel x86 processor, referred to as x86-64</li> </ul>			

# Intel x86 Processors

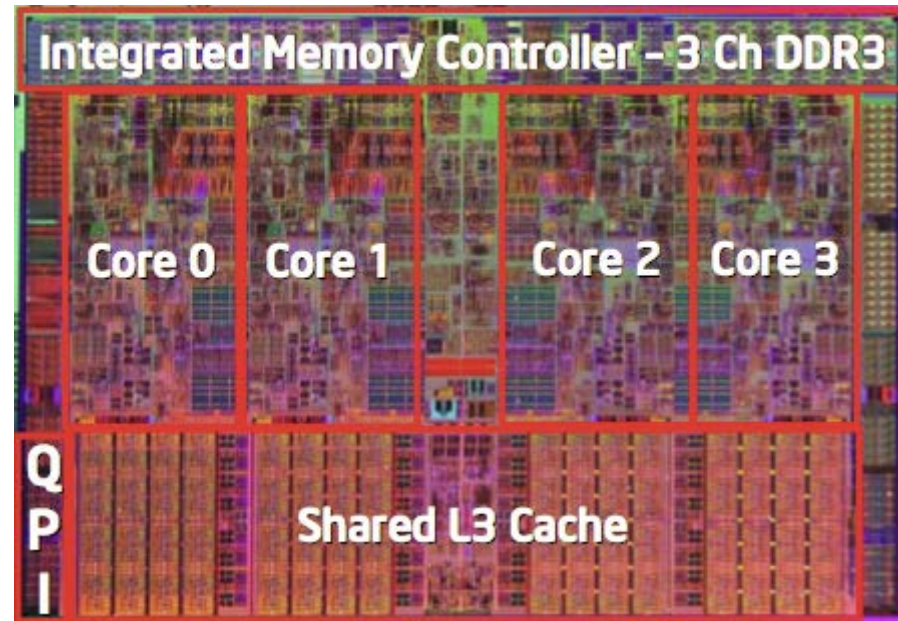
## ■ Machine Evolution

■ 486	1989	1.9M
■ Pentium	1993	3.1M
■ Pentium/MMX	1997	4.5M
■ PentiumPro	1995	6.5M
■ Pentium III	1999	8.2M
■ Pentium 4	2001	42M
■ Core 2 Duo	2006	291M
■ Core i7	2008	731M

## ■ Added Features

- Instructions to support multimedia operations
  - Parallel operations on 1, 2, and 4-byte data
- Instructions to enable more efficient conditional operations
- More cores!

## Intel Core i7



# More information

## ■ References for Intel processor specifications:

- Intel's "automated relational knowledgebase":
  - <http://ark.intel.com/>
- Wikipedia:
  - [http://en.wikipedia.org/wiki/List\\_of\\_Intel\\_microprocessors](http://en.wikipedia.org/wiki/List_of_Intel_microprocessors)

# x86 Clones: Advanced Micro Devices (AMD)

## ■ Historically

- AMD has followed just behind Intel
- A little bit slower, a lot cheaper

## ■ Then

- Recruited top circuit designers from Digital Equipment and other downward trending companies
- Built Opteron: tough competitor to Pentium 4
- Developed x86-64, their own extension of x86 to 64 bits

# Our Coverage

- **IA32**

- The traditional x86

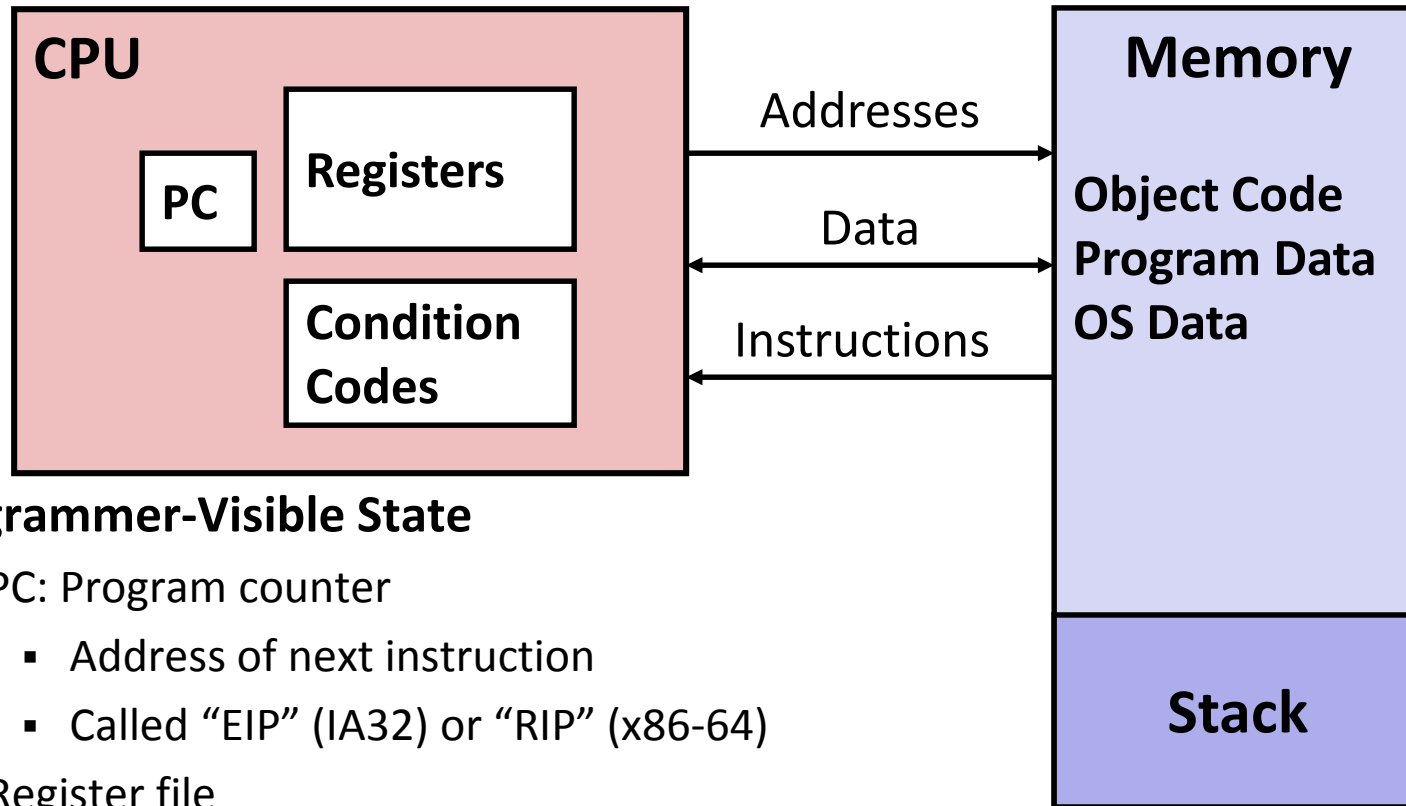
- **x86-64**

- The emerging standard – all lab assignments use x86-64!

# Definitions

- **Architecture:** (also instruction set architecture or ISA)  
The parts of a processor design that one needs to understand to write assembly code
  - “What is directly visible to software”
- **Microarchitecture:** Implementation of the architecture
- Is cache size “architecture”?
- How about core frequency?
- And number of registers?

# Assembly Programmer's View



## ■ Programmer-Visible State

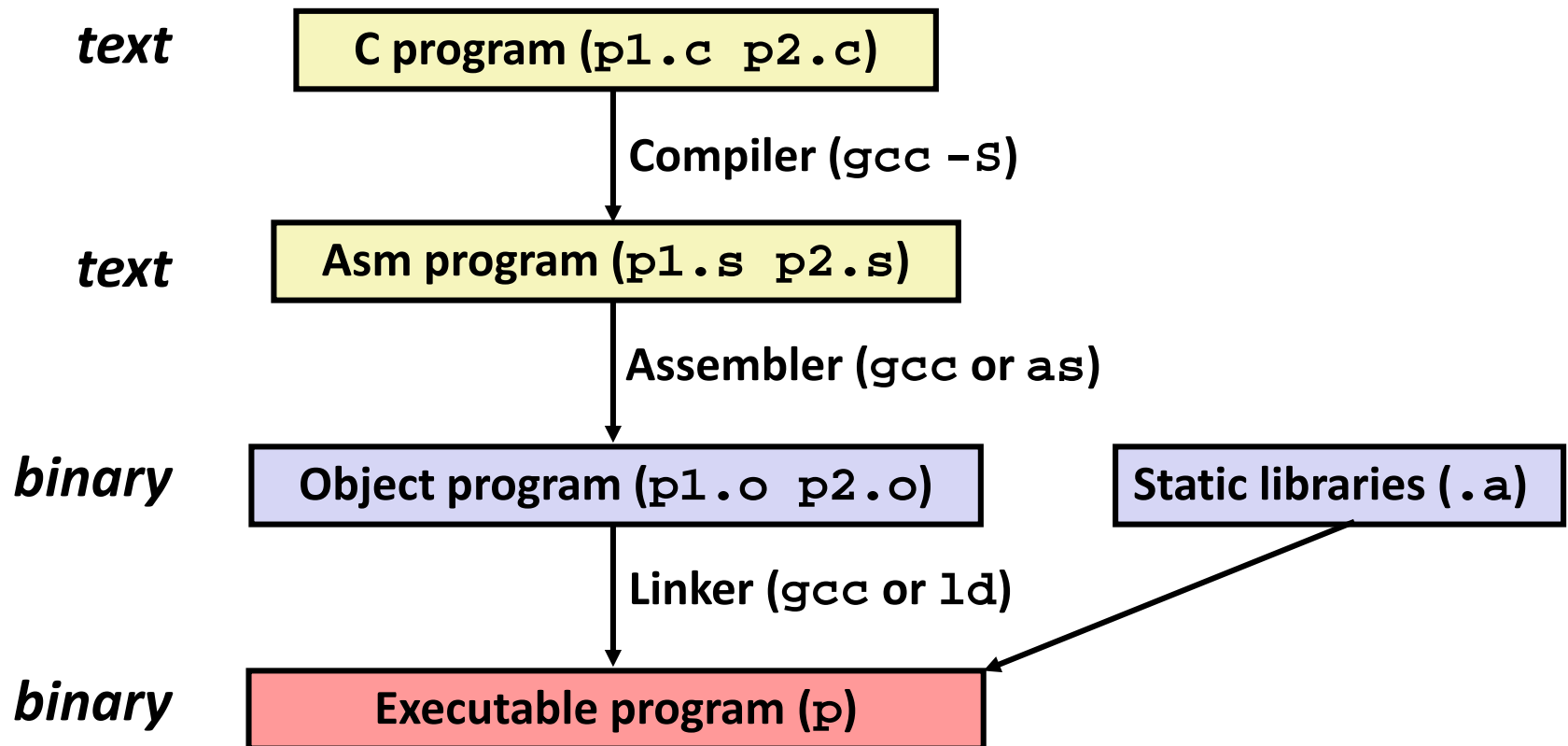
- PC: Program counter
  - Address of next instruction
  - Called “EIP” (IA32) or “RIP” (x86-64)
- Register file
  - Heavily used program data
- Condition codes
  - Store status information about most recent arithmetic operation
  - Used for conditional branching

## ■ Memory

- Byte addressable array
- Code, user data, (some) OS data
- Includes stack used to support procedures (we’ll come back to that)

# Turning C into Object Code

- Code in files `p1.c` `p2.c`
- Compile with command: `gcc -O1 p1.c p2.c -o p`
  - Use basic optimizations (`-O1`)
  - Put resulting binary in file `p`



# Compiling Into Assembly

## C Code

```
int sum(int x, int y)
{
    int t = x+y;
    return t;
}
```

## Generated IA32 Assembly

```
sum:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    addl 8(%ebp),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Obtain with command

```
gcc -O1 -S code.c
```

Produces file `code.s`

# Three Basic Kinds of Instructions

- **Perform arithmetic function on register or memory data**
- **Transfer data between memory and register**
  - Load data from memory into register
  - Store register data into memory
- **Transfer control**
  - Unconditional jumps to/from procedures
  - Conditional branches

# Assembly Characteristics: Data Types

- **“Integer” data of 1, 2, 4 (IA32), or 8 (just in x86-64) bytes**
  - Data values
  - Addresses (untyped pointers)
- **Floating point data of 4, 8, or 10 bytes**
- **What about “aggregate” types such as arrays or structs?**
  - No aggregate types, just contiguously allocated bytes in memory

# Object Code

## Code for `sum`

0x401040 <sum>:

0x55

0x89

0xe5

0x8b

0x45

0x0c

0x03

0x45

0x08

0x89

0xec

0x5d

0xc3

- Total of 13 bytes
- Each instruction 1, 2, or 3 bytes
- Starts at address 0x401040
- Not at all obvious where each instruction starts and ends

## ■ Assembler

- Translates `.s` into `.o`
- Binary encoding of each instruction
- Nearly-complete image of executable code
- Missing links between code in different files

## ■ Linker

- Resolves references between object files and (re)locates their data
- Combines with static run-time libraries
  - E.g., code for `malloc`, `printf`
- Some libraries are *dynamically linked*
  - Linking occurs when program begins execution

# Machine Instruction Example

```
int t = x+y;
```

```
addl 8(%ebp),%eax
```

Similar to expression:

```
x += y
```

More precisely:

```
int eax;
```

```
int *ebp;
```

```
eax += ebp[2]
```

```
0x401046:    03 45 08
```

- **C Code:** add two signed integers

- **Assembly**

- Add two 4-byte integers
  - “Long” words in GCC speak
  - Same instruction whether signed or unsigned

- Operands:

**x:** Register      **%eax**

**y:** Memory      **M[%ebp+8]**

**t:** Register      **%eax**

–Return function value in **%eax**

- **Object Code**

- 3-byte instruction
- Stored at address **0x401046**

# Disassembling Object Code

## Disassembled

```

00401040 <_sum>:
  0:      55          push   %ebp
  1:      89 e5       mov    %esp,%ebp
  3:      8b 45 0c    mov    0xc(%ebp),%eax
  6:      03 45 08    add   0x8(%ebp),%eax
  9:      89 ec       mov    %ebp,%esp
  b:      5d          pop    %ebp
  c:      c3          ret

```

## ■ Disassembler

`objdump -d p`

- Useful tool for examining object code (`man 1 objdump`)
- Analyzes bit pattern of series of instructions (delineates instructions)
- Produces near-exact rendition of assembly code
- Can be run on either `p` (complete executable) or `p1.o / p2.o` file

# Alternate Disassembly

## Object

```
0x401040:
  0x55
  0x89
  0xe5
  0x8b
  0x45
  0x0c
  0x03
  0x45
  0x08
  0x89
  0xec
  0x5d
  0xc3
```

## Disassembled

```
0x401040 <sum>:      push    %ebp
0x401041 <sum+1>:      mov     %esp,%ebp
0x401043 <sum+3>:      mov     0xc(%ebp),%eax
0x401046 <sum+6>:      add    0x8(%ebp),%eax
0x401049 <sum+9>:      mov     %ebp,%esp
0x40104b <sum+11>:   pop     %ebp
0x40104c <sum+12>:   ret
```

### ■ Within gdb debugger

```
gdb p
```

```
disassemble sum
```

```
(disassemble function)
```

```
x/13b sum
```

```
(examine the 13 bytes starting at sum)
```

# What Can be Disassembled?

```
% objdump -d WINWORD.EXE

WINWORD.EXE:      file format pei-i386

No symbols in "WINWORD.EXE".
Disassembly of section .text:

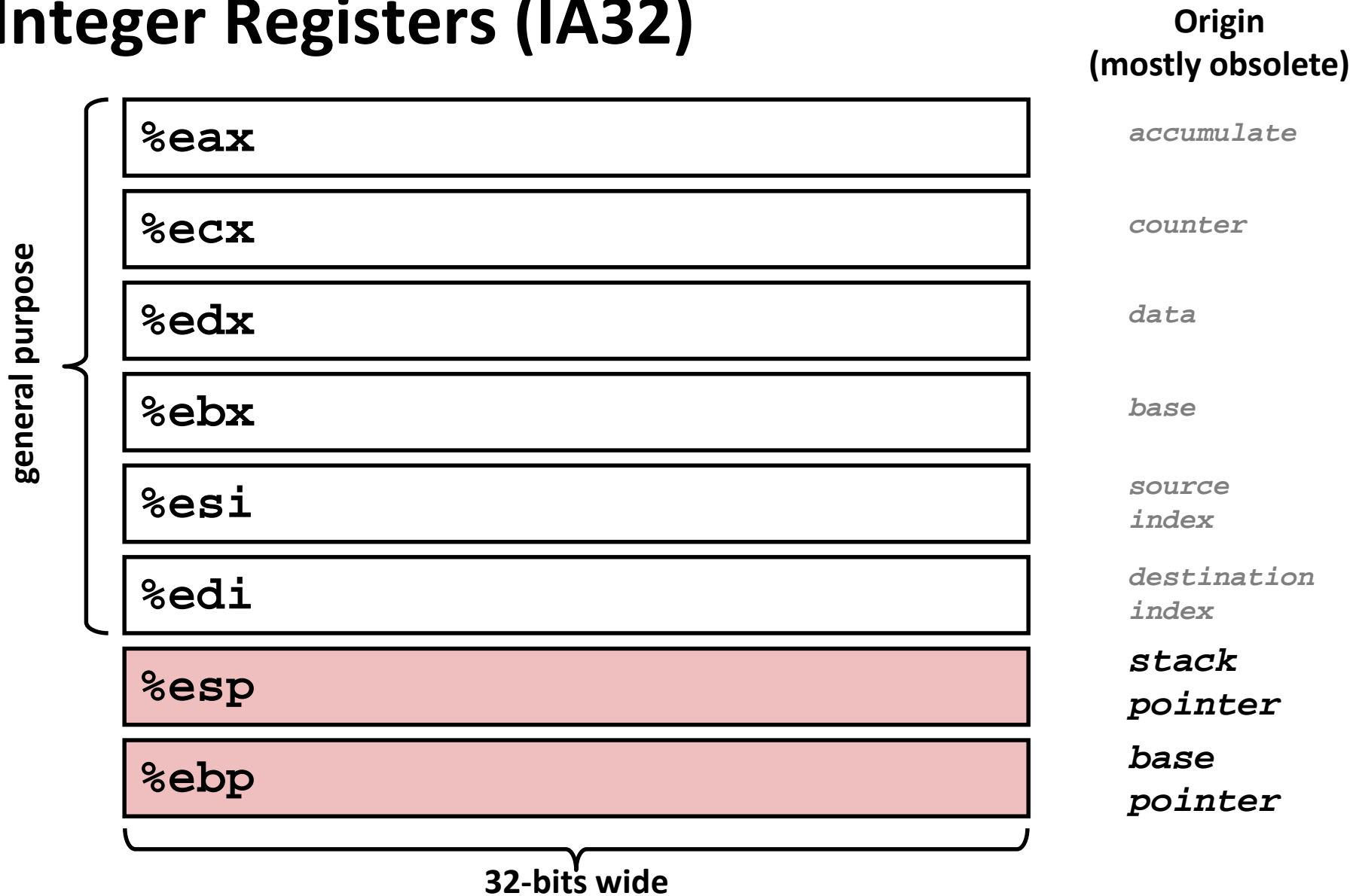
30001000 <.text>:
30001000:  55                push    %ebp
30001001:  8b ec            mov     %esp,%ebp
30001003:  6a ff            push   $0xffffffff
30001005:  68 90 10 00 30   push   $0x30001090
3000100a:  68 91 dc 4c 30   push   $0x304cdc91
```

- Anything that can be interpreted as executable code
- Disassembler examines bytes and reconstructs assembly source

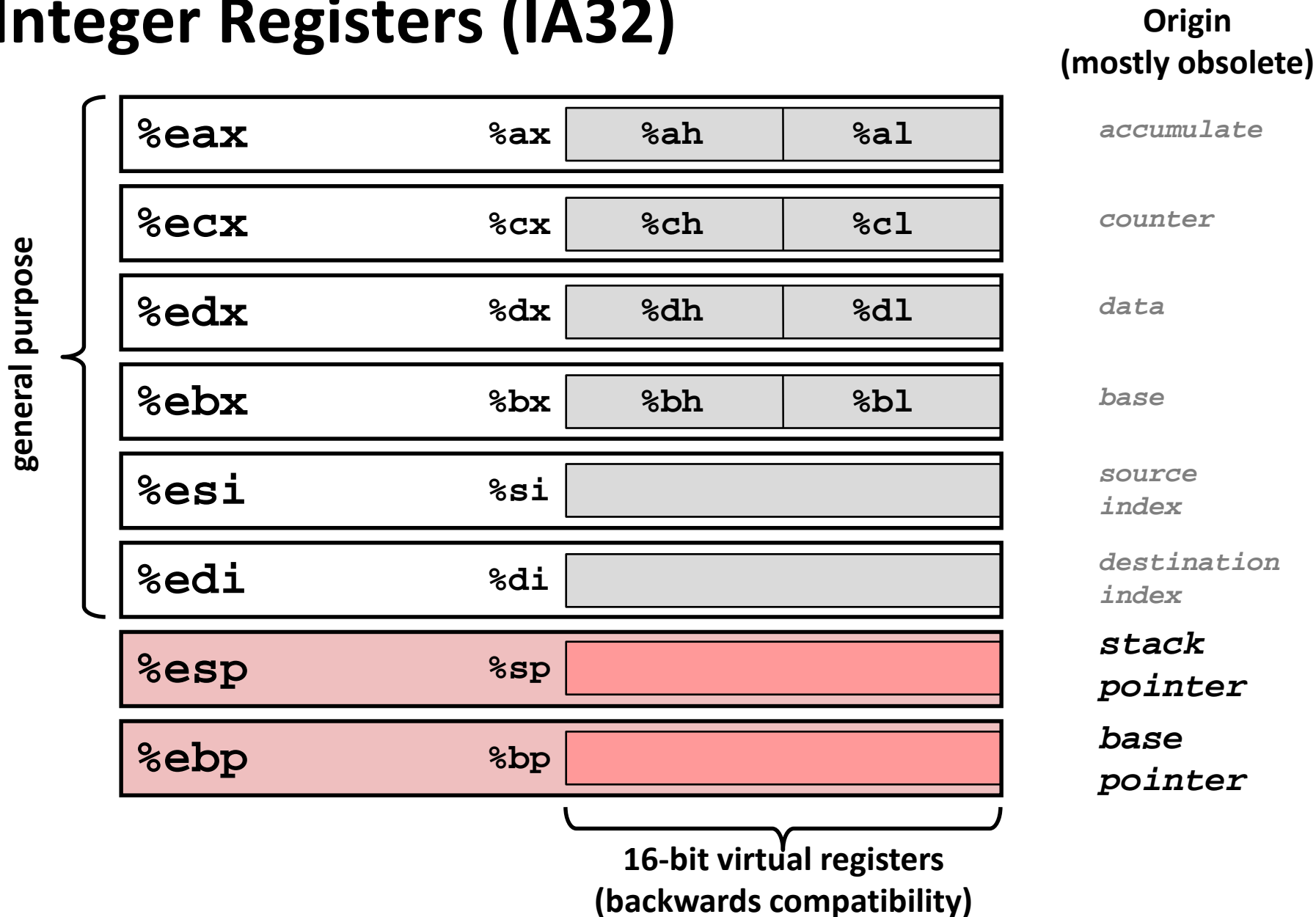
# What Is A Register?

- A location in the CPU that stores a small amount of data, which can be accessed very quickly (once every clock cycle)
- Registers are at the heart of assembly programming
  - They are a precious commodity in all architectures, but *especially* x86

# Integer Registers (IA32)



# Integer Registers (IA32)



# x86-64 Integer Registers

<code>%rax</code>	<code>%eax</code>
<code>%rbx</code>	<code>%ebx</code>
<code>%rcx</code>	<code>%ecx</code>
<code>%rdx</code>	<code>%edx</code>
<code>%rsi</code>	<code>%esi</code>
<code>%rdi</code>	<code>%edi</code>
<code>%rsp</code>	<code>%esp</code>
<code>%rbp</code>	<code>%ebp</code>

64-bits wide

<code>%r8</code>	<code>%r8d</code>
<code>%r9</code>	<code>%r9d</code>
<code>%r10</code>	<code>%r10d</code>
<code>%r11</code>	<code>%r11d</code>
<code>%r12</code>	<code>%r12d</code>
<code>%r13</code>	<code>%r13d</code>
<code>%r14</code>	<code>%r14d</code>
<code>%r15</code>	<code>%r15d</code>

- Extend existing registers, and add 8 new ones; *all* accessible as 8, 16, 32, 64 bits.

# Summary: Machine Programming

- **What is an ISA (Instruction Set Architecture)?**
  - Defines the system's state and instructions that are available to the software
- **History of Intel processors and architectures**
  - Evolutionary design leads to many quirks and artifacts
- **C, assembly, machine code**
  - Compiler must transform statements, expressions, procedures into low-level instruction sequences
- **x86 registers**
  - Very limited number
  - Not all general-purpose